Presented at Technology 2002:
The Third National Technology Transfer Conference and Exposition,
Baltimore, December, 1992.

Tree Classification Software

Wray Buntine, RIACS
NASA Ames Research Center
Mail Stop 269-2
Moffet Field, CA 94035

ABSTRACT

This paper introduces the IND Tree Package to prospective users. IND does supervised learning using classification trees. This learning task is a basic tool used in the development of diagnosis, monitoring and expert systems. IND was developed as part of a NASA project to semi-automate the development of data analysis and modelling algorithms using artificial intelligence techniques. IND integrates features from Breiman *et al.*'s CART and Quinlan's C4.5 with newer Bayesian and minimum encoding methods for growing classification trees and graphs. IND also provides an experimental control suite on top. The newer features give improved probability estimates often required in diagnostic and screening tasks. The package comes with a manual, Unix ``man'' entries, and a guide to tree methods and research. IND is implemented in C under Unix, and has been beta-tested at university and commercial research laboratories in the United States.

DIAGNOSIS AND CLASSIFICATION

A common inference task is where we learn to make a discrete prediction about some case given other details about the case. For instance, in financial credit assessment we wish to decide whether to accept or reject a customer's application for a loan given particular personal information. In monitoring a subsystem of the space shuttle, measurements such as flow rates and temperature are continuously recorded and we need to screen those measurements to decide if the system is in normal or abnormal operation. If the system is in abnormal operation we might further wish to try and predict the type of abnormality present. This prediction task is the basic task of many expert systems, health monitoring systems, diagnostic systems, etc. Furthermore, more complex problems can often be broken down into a sequence of simple prediction problems. For instance, speech understanding, converting the spoken word into written text, is a sequence of prediction tasks about each phoneme.

In medical diagnosis, or diagnosis of equipment subsystems, we need more than just a prediction, we need a careful probabilistic assessment. A simplistic medical example will bring this point home. Suppose your doctor suspects you have a cyst in your abdomen. The options (1 or 2) and outcomes (A or B) give the following set of possibilities: (1A) operates, discovers a cyst, removes it, and you're grateful; (1B) operates, no cyst found, but you're left with the medical bill and a day recovery in hospital; (2A), doesn't operate but the cyst exists and causes medical complications due to lack of treatment; (2B), doesn't operate, no cyst exists. Each case has important implications to you both financially and in quality of life. With a careful probabilistic assessment of the existence of a cyst, you can weigh up the options and decide which option (1 or 2) is the most beneficial to you. For instance, if the medical bill is insignificant compared to the potential medical complications, then you would decide to have the operation even if there was a small chance of having the cyst. If the potential medical complications were insignificant, you would only decide to operate if there was a very high probability of having the cyst. This process of decision analysis requires as input probabilities about the new case in question.

In health monitoring and diagnosis, these probability assessments are needed when the system is being used to screen cases, i.e. the computer systems scans the on-line monitoring data and at certain time points alerts a human expert that a potentially anomalous situation has arisen. Probability assessments such as the "probability of equipment failure" can be used to determine which of the many cases scanned should be forwarded to the human expert for the more costly process of manual inspection.

I will refer to this prediction problem as *classification*, where the aim is to *classify* each new case. One common technique for developing a system to do prediction or probability assessment about new cases is to examine a database of cases, for instance collected historically. Assume that hindsight tells us which is the correct classification for each case in the data base, so for each we know which prediction was optimal. From the data base we use statistical techniques to "discover" or "learn" how to do the predictions for new unseen cases. This learning technique is represented in Figure 1. The process requires three main forms of input: an *expert* who is able to advise on the problem, help configure the system, etc., a *data base* of correctly classified cases to use in the learning process, and a *model family* from which the learning algorithm is to select a "good" model for doing prediction or probabilistic assessment.

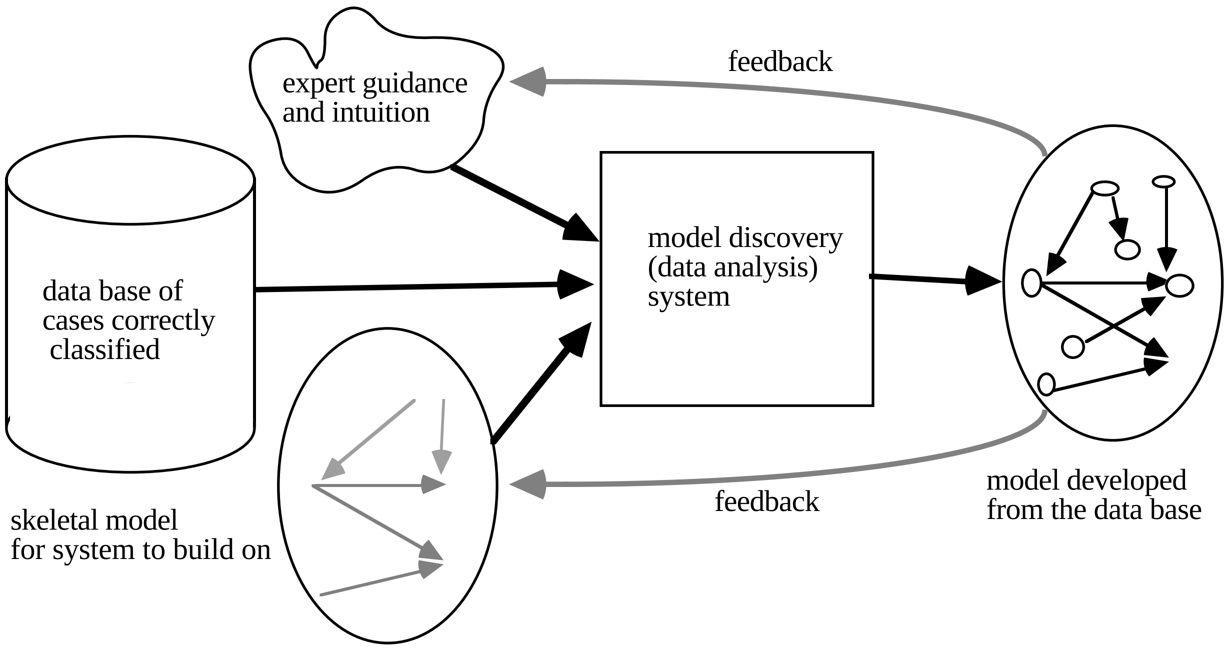

feedback
expert guidance and intuition
data base of cases correctly classified
model discovery (data analysis) system
skeletal model for system to build on
feedback
model developed from the data base

Figure 1. Learning prediction models from data.

This model learning or discovery process is a useful technique in almost every industry, finance, manufacturing, etc., wherever on-line databases are stored and important predictions have to be made on a regular basis about new cases before they enter the data base. Not surprisingly, there are many different fields of science that address this problem as one of their central concerns. In artificial intelligence it is referred to as the classification or induction problem. Techniques include tree and rule learning algorithms of the form I will present in this paper. In statistics it is referred to as the discrimination problem, and common techniques are the linear models used in the finance and banking industry for credit assessment. In pattern recognition it is referred to as supervised learning. In neural networks it is the classification and generalization problem and is routinely investigated using a number of network architectures. These diverse fields are all studying the same problem, "learning to predict", and present a confusing array of methodologies and paradigms for addressing that problem. They differ in the following aspects:

Model family: Which class of models are being used to do prediction? In Figure 1 this corresponds to the "skeletal model". I present classification tree and classification graph model families in this paper.

Statistical philosophy: How is learning to occur? That is, what statistical principles if any are used to develop the central box in Figure 1?

Computational and optimization methods: What are the basic computational methods used in terms of efficiency, optimality, search method, etc.?

Methodological support: What methodology does the analyst use to go about applying the technique to a real problem? For statisticians this is the "consultancy phase" rarely covered in university courses. In artificial intelligence this is the process of "knowledge engineering".

I will refer to the general task of learning how to predict (or estimate probabilities) from data as the classification task. The next section discusses the design of tools for this task. After this, the model family considered in this paper is addressed, and the IND program presented.

This research is part of a broader effort to semi-automate the development of classification algorithms. The goal of this research is to develop generic tools for learning from data and from partial models of the domain, and to develop the capability to rapidly develop and tailor these learning tools for particular domains given, for instance, specification of the kinds of models that are of interest. When encountering a new application, we sometimes find that off-the-shelf-tools, such as IND, need some modification in order to better suit the task. A good development methodology lets that be done with minimum fuss.

Rather than following a particular field, our group takes a multidisciplinary approach and combine a range of methods required to address the classification task. Our group uses artificial intelligence search techniques for discrete search problems, and standard numerical techniques for continuous problems. We use some of the flexible knowledge representation schemes from artificial intelligence as skeletal models or model families (see Figure 1), and use Bayesian statistical and decision methods for the statistical philosophy underlying our learning algorithms. This methodology allows rapid development of approximately optimal algorithms and so avoids the many pitfalls of *ad hoc* development according to "hunches" and the time-consuming refinement cycle that this entails. This theoretical framework of "statistical philosophy" plus "optimization methods" is important because empirical, *ad hoc* development of algorithms in neural networks and early machine learning has been time consuming and is often plagued by unexplained problems. Empirical validation of our algorithms is also important to check approximations made in interpreting the Bayesian theory. We do this empirical validation by applying the algorithms to a battery of recognized learning problems taken from the literature, or manufactured problems. A summary of our groups development methodology is given in Figure 2. This has lead to the development of a number of sophisticated algorithms, one of which was the Autoclass system, show-cased at Technology 2001 by Stutz, Cheeseman, and Taylor at San Jose, December 1991.

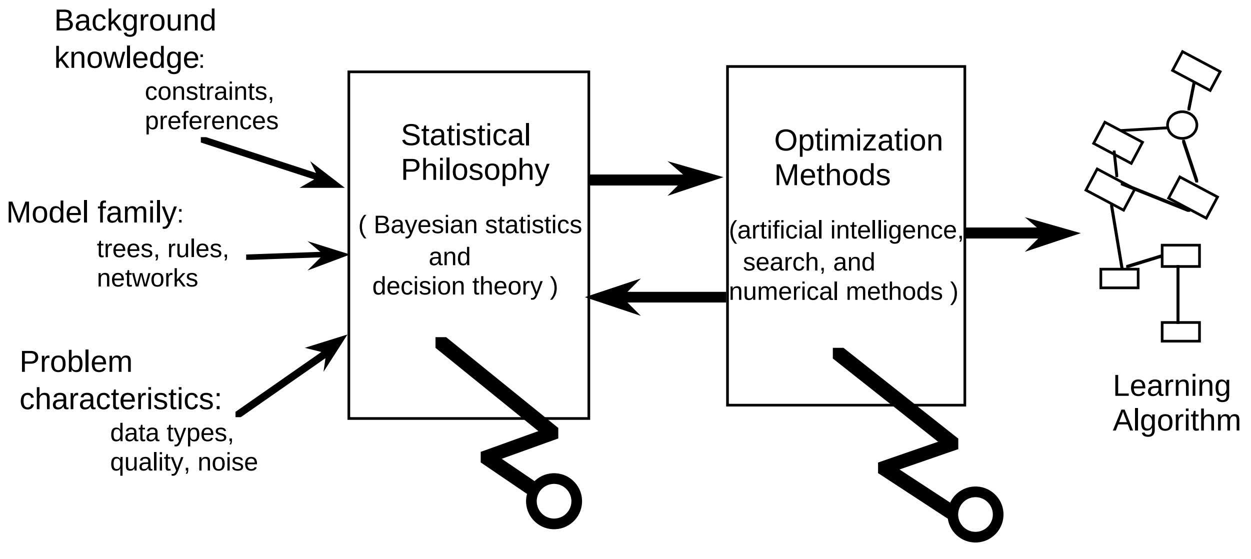


Figure 2. Semi-automatic development of learning algorithms

The justification for Bayesian decision theory, used in the first box in Figure 2, comes from fundamental principles of how uncertain reasoning should be done [1]. The theory applies widely in inference and plausible reasoning and its use is continually expanding in artificial intelligence and neural networks. But there is not a single "Bayesian learning algorithm," as some people mistakenly believe when they learn about the simple Bayesian classifiers developed in pattern recognition. Rather, Bayesian decision theory presents computational guidelines on how learning should be done for many different learning problems, and shows how to tailor methods to particular applications. This means our algorithms can be fine-tuned to the requirements of an individual application. IND has some basic features that allow such tuning.

The IND package described later does prediction using *decision trees* or *decision graphs* and does probability evaluation using *class probability trees* or *graphs.* These are a general form of classification rule that mix discrete and continuous data and are often suited when there is believed to be some form of non-linear structure in the data. A decision tree is shown in Figure 3b. This has the classes *hypo* (hypothyroid) and *not* (not hypothyroid) at the leaves. This tree is for a *two-class classification problem* because there are two different classes that leaf nodes recommend. This tree is processed as follows. Look at the new case you wish to evaluate. Is its value of *TSH* greater than 200? If so take the left branch of the tree and you have reached a leaf. The tree says to predict *hypo*, i.e. hypothyroid. If however the value of *TSH* was less than 200, then take the right branch. Now you have a subtree and you repeat the process. In this case is *Pregnant* true or not? This tree is referred to as a "decision tree" because decisions about class membership are represented at the leaf nodes. Notice that the real valued attributes *TSH* has been incorporated into the tree by making a binary test of the form "attribute < cut-point". Also, the tree need not be binary; if a 4-valued attribute is tested at one of the nodes, then the tree might have 4 branches coming from the node, one for each value.

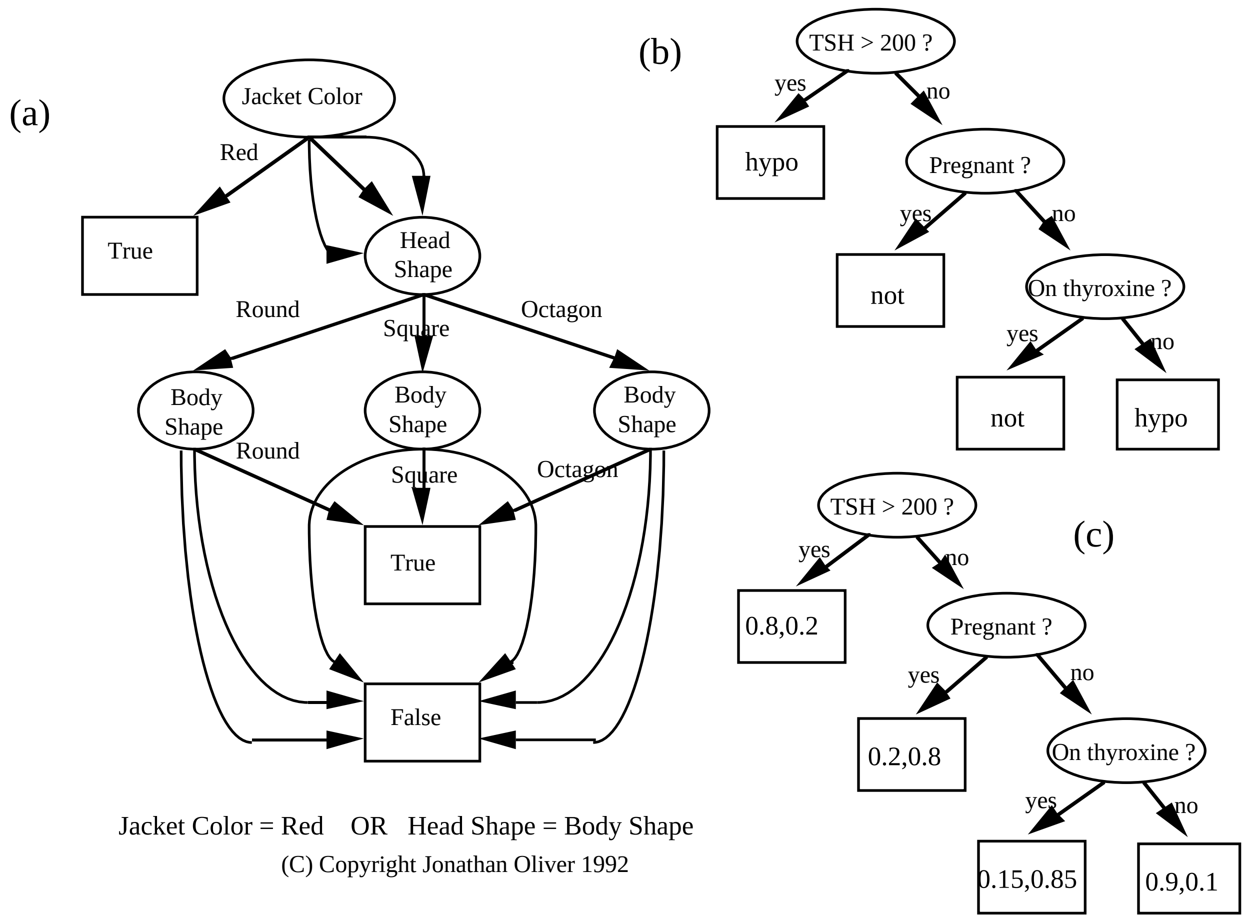


Figure 3. (a) This is a decision graph for the boolean problem given in the figure. Start at the root at trace through the graph to arrive a decision. (b) This is a decision tree for the "hypothyroid" application. (c) This is a class probability tree. Leaf nodes give estimates of class probability.

In typical problems involving noise, class probabilities are usually given at the leaf nodes instead of class decisions, forming a *class probability tree* (where each leaf node has a vector of class probabilities). A corresponding class probability tree is given in Figure 3c. The leaf nodes give predicted probabilities for the two classes. Notice that this tree is a representation for a conditional probability distribution of class given information higher in the tree. This is the statistical interpretation of the tree that Bayesian methods use in developing a learning algorithm.

Methods for learning decision trees and class probability trees have been under development in some form or another for some two decades. The standard technique for building classification trees from data is the so-called recursive partitioning algorithm that forms the basis of systems such as Quinlan's ID3 and C4.5 [2,3], well know in the machine learning literature, and Breiman, Friedman, Olshen and Stone's CART [4], well known in the applied statistics literature. These methods are largely reimplemented in IND.

A more complex structure is shown in Figure 3a. This is a decision graph, and it is also for a two-class problem. Graphs and trees can also be applied to problems with three or more classes. The graph is processed in exactly the same way as a decision tree, however notice that the graph allows more general connections. This graph represents the boolean function "*jacket-color* = red or *head-shape* = *body-shape*". This function would take a complex tree to represent. With graphs we can represent concepts more efficiently. Methods for learning decision graphs and class probability graphs have only recently appeared, and they supersede trees in that they are a more general representation. IND version 2.1 includes experimental versions of these methods coded up by Jon Oliver [7].

AN OVERVIEW OF THE IND PACKAGE

IND is a suite of C programs and C shell scripts for building tree classifiers and graph classifiers of the kind just described. Currently, several different methods are integrated (CART style; the regression aspect of CART is not implemented, C4.5 style, MML/MDL, and Bayesian averaging). Careful checking has been done so that IND reimplements CART and C4.5 fairly faithfully.. The new Bayesian/MML/MDL features can give performance improvement over these in many cases when used appropriately.

IND can be operated in a variety of modes that allow the novice to build trees without too much fuss, and also allow the expert to fine tune the algorithms to particular applications. If you're interested in applying IND to applications, advice is given in the manual on which options to use and how to take into account features of your application and data when configuring your use of IND. If you're interested in running comparative trials or just experimenting with tree software, IND provides extensive experimental control (random partitioning, cross validation) and significance testing. The code for IND is provided (and sometimes even moderately documented) so you can develop your own extensions.

The IND Manual: "An Introduction to IND and Recursive Partitioning" is the best place to start if you are unfamiliar with IND or recursive partitioning. The manual contains an introduction to IND that walks through a few typical sessions, a tutorial on recursive partitioning, a description of IND options, and a fairly complete glossary and bibliography. The is an enormous literature on decision trees and their applications so the manual also contains a brief guide to the literature.

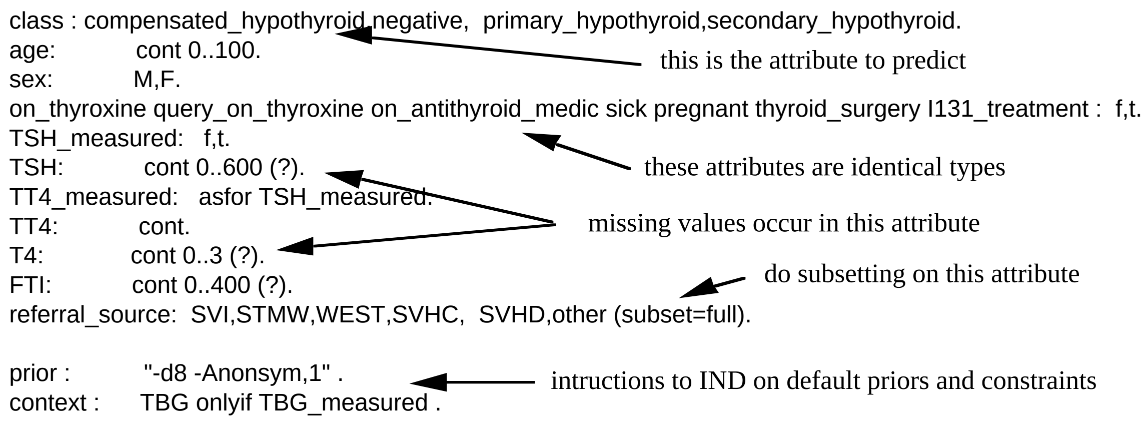
IND has a variety of features including: interactive control of tree building, variable search such as multi-ply look-ahead, missing values and subsetting, handling of utilities and cost functions, prediction of error rates and utilities, a range of priors for the Bayesian methods, printing options, a classifier, etc., a user manual, and a start of the art guide to tree learning research. IND has the look and feel of a typical Unix system and comes with "man" entries. The system has been developed exclusively in a SUN workstation environment under various releases of SunOS UNIX. It compiles under Kernighan and Ritchie C, `cc` and `gcc`, although in future will be converted to ANSI standard C. Various users have ported the system to HP, IBM and other Unix platforms and their changes have been incorporated in the latest release.

In November 1991 the IND Tree Package version 1.0 was prepared and released as a beta test to the research and development community. About 40 universities and R&D laboratories in the US currently have the beta test code. The code release includes 200 pages of documentation and 15000 lines of C code and C shell scripts. The code has had three minor revisions since version 1.0. Version 2.0 is being prepared for release though COSMIC, and should be submitted October 1992. Version 2.0 includes extensions to the user interface and fixes all the bugs reported on the beta-test, but does not contain the decision graph routines. Version 2.1 includes algorithms for learning decision graphs, a reimplementation of C4.5, and sophisticated any-time search algorithms for returning better quality trees and graphs. Version 2.1 was released as beta test in January 1993.

Main use of the code to date has been in bench-marking, comparative studies, and comparative research on related algorithms, although groups in several different commercial and scientific areas currently have the code. Comparative studies done by several international research groups have found the code to be a good implementation, somewhat slower than the original CART code, but more flexible, and easier to use. The new Bayesian extensions have also been found to give significant improvement over earlier tree algorithms, particularly in providing probability estimates, an important task for diagnosis and monitoring.

MODULES IN THE IND PACKAGE

The first task in using IND is to format your data into an appropriate text file and run it through the data conversion routine in IND. The routine `encsmpl` will produce an IND data description file for you, see Figure 4, and encode the data into IND's internal format. This data description file can then be modified to add defaults, utilities, constraints, etc., to configure IND for this data. An extract of a text file matching the description file in Figure 4 is given in Figure 5.



```
class : compensated_hypothyroid,negative,  primary_hypothyroid,secondary_hypothyroid.
age:           cont 0..100.
sex:           M,F.
on_thyroxine query_on_thyroxine on_antithyroid_medic sick pregnant thyroid_surgery I131_treatment :  f,t.
TSH_measured:   f,t.
TSH:           cont 0..600 (?).
TT4_measured:   asfor TSH_measured.
TT4:           cont.
T4:            cont 0..3 (?).
FTI:           cont 0..400 (?).
referral_source:  SVI,STMW,WEST,SVHC,  SVHD,other (subset=full).

prior :         "-d8 -Anonsym,1" .
context :      TBG onlyif TBG_measured .
```

this is the attribute to predict
these attributes are identical types
missing values occur in this attribute
do subsetting on this attribute
intructions to IND on default priors and constraints

Figure 4. The data description file input to IND.

```
negative 36 F f f f f f f f t 0.22 t 191 0.98 194 other
negative 73 F f f f t f f f f ? t 119 0.92 129 SVI
compensated_hypothyroid 34 F f f f f t f f t 19 146 ? 125 other
```

Figure 5. Sample input data file matching description in Figure 4.

Once IND has the data encoded, IND can be operated at a number of different levels, depending on the requirements of the user. The simplest level is to use commands that have general prepared styles for tree generation. The command `mktree` shown in the top of Figure 6 uses prepared styles to drive the basic tree generation, pruning and classification routines. A sample run from `mktree` is given in Figure 7 at the end of the paper. This used the verbose option to automatically explain each component of IND and how it was configured. More experienced users of IND may like to make better use of the range of features. To do this, the low level routines can be called directly. All routines are controlled using the data description format of Figure 4 together with standard Unix style command options. Users may also wish to perform cross-validation to estimate error rates, or run experiments using a number of different tree styles to help in configuring IND for their problem. This can be done using the `ttest` utility shown at the top of Figure 6. This utility collates statistics required for you to analyze each run.

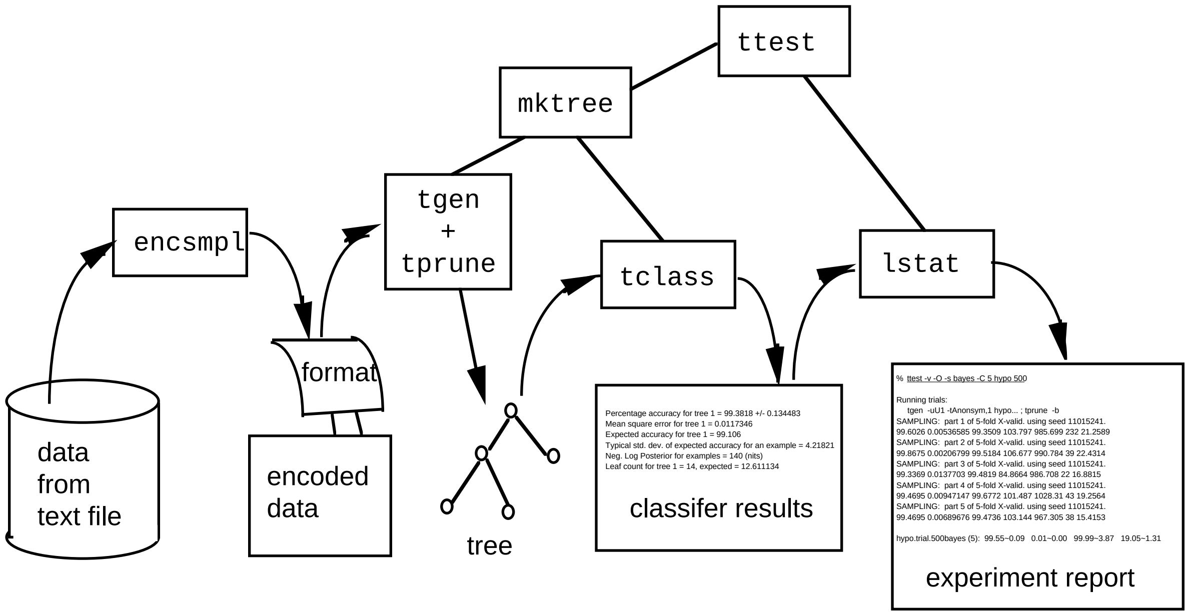
Some of the prepared styles available for the novice user of IND are as follows:

bayes, **mml** : The simple.bayes style is useful when you know that most of the attributes supplied are relevant and that moderate accuracy is achievable. The mml style assumes poorer attribute quality. Both styles use Bayesian smoothing.

cart : A number of variations of basic CART are reimplemented in IND, although multivariate splits and surrogate splits are not implemented. Basic cart style using subsetting, twoing, cross validation cost complexity pruning and a simple stopping rule.

c4 : A style like C4.5 is implemented with subsetting (different to Quinlan's original), C4.5 pruning and the gain ratio splitting rule.

dgraph : Build a decision graph using the mml style above.



```
Percentage accuracy for tree 1 = 99.3818 +/- 0.134483
Mean square error for tree 1 = 0.0117346
Expected accuracy for tree 1 = 99.106
Typical std. dev. of expected accuracy for an example = 4.21821
Neg. Log Posterior for examples = 140 (nits)
Leaf count for tree 1 = 14, expected = 12.611134
```

```
%  ttest -v -O -s bayes -C 5 hypo 500

Running trials:
    tgen  -uU1 -tAnonsym,1 hypo... ; tprune  -b
SAMPLING:  part 1 of 5-fold X-valid. using seed 11015241.
99.6026 0.00536585 99.3509 103.797 985.699 232 21.2589
SAMPLING:  part 2 of 5-fold X-valid. using seed 11015241.
99.8675 0.00206799 99.5184 106.677 990.784 39 22.4314
SAMPLING:  part 3 of 5-fold X-valid. using seed 11015241.
99.3369 0.0137703 99.4819 84.8664 986.708 22 16.8815
SAMPLING:  part 4 of 5-fold X-valid. using seed 11015241.
99.4695 0.00947147 99.6772 101.487 1028.31 43 19.2564
SAMPLING:  part 5 of 5-fold X-valid. using seed 11015241.
99.4695 0.00689676 99.4736 103.144 967.305 38 15.4153

hypo.trial.500bayes (5):  99.55~0.09   0.01~0.00   99.99~3.87   19.05~1.31
```

Figure 6. Overview of the modules in IND.

CLASS PROBABILITY TREE THEORY

In this section I briefly review the Bayesian theory of learning classification trees. This theoretical section should be skipped if your interests lie in applications of the algorithm. The section introduces the theory behind the unique Bayesian aspects of the IND package. More details of this theory are given in [5,6]. An excellent introduction to tree methods can be found in [2]. Theory behind the graph components of IND available in beta-test version 2.1 can be found in [7]. The methods discussed here are developed according to the algorithm design strategy presented in the earlier design section.

The basic tenet of Bayesian decision theory is that if we do not know something with reasonable certainty, then we should look at some reasonable and mutually exclusive alternatives and weigh them up, to help us make a "representative" decision. A reasonable alternative is one we currently have high subjective belief in. I will explain how this applies to trees, based on material in [6]. The formulation is sufficiently general so that it could just as well be applied to other classification models such as probabilistic rules, Bayesian networks, or one of many other knowledge representations from artificial intelligence, neural networks or statistics that have a probabilistic interpretation.

Class probability trees have a vector of class probabilities at their leaves, as shown in Figure 3c. They represent a conditional probability distribution of class value conditioned on other details about the case. A particular class probability tree can be represented by its discrete component *T*, the *tree structure* given by the shape of the tree and the tests at the leaves, and its continuous component *S*, the leaf class probabilities. This gives the conditional probability distribution *Pr*(*class*|*case*,*T*,*S*), which is the *likelihood function* for a classified case (*class*,*case*) using the class probability tree specified by *T* and *S*.

Suppose we are given a training sample *Sample* consisting of classified cases *cases* and their classes *classes*, together with a new case, *new-case*, whose class, *new-class*, we wish to predict. If the goal is to minimize errors in prediction (other utility functions can be handled similarly), decision theory says we should choose the class *new-class* to maximize the posterior class probability *Pr*(*new-class*| *new-case*, *Sample*). Using the tree model, this expression can be expanded using the laws of probability theory to obtain the posterior average of the class probabilities predicted for *new-class* from all possible class probability trees:

$$\begin{aligned} Pr(\textit{new-class}\,|\,\textit{new-case}, \textit{Sample}) &= \sum_T \int_S Pr(\textit{new-class}\,|\,\textit{new-case}, T, S)\, Pr(T,S \mid \textit{Sample})\, dS \\ &= \sum_T Pr(\textit{new-class}\,|\,\textit{new-case}, T, \textit{Sample})\, Pr(T \mid \textit{Sample}) \qquad (1) \end{aligned}$$

where the summations are over the space of all possible tree structures *T*, and

$$Pr(T \mid Sample) \text{ proportional-to } \int_S Pr(classes \mid cases, T, S)\, Pr(S \mid T)\, Pr(T)\, dS$$

$$Pr(new\text{-}class \mid new\text{-}case, T, Sample) \text{ proportional-to } \int_S Pr(new\text{-}class \mid new\text{-}case, T, S)\, Pr(S \mid T, Sample)\, dS$$

Formula (1) simply says to average the class predictions made for each tree. That is, since we aren't certain which tree is "true", we hedge our bets over reasonable trees. The *posterior* probability of the tree structure T, $Pr(T \mid Sample)$, is the weight used in the averaging process. The probabilities appearing in the formula above are calculated in log-space, to prevent underflow, and are sometimes referred to as "code-lengths" (because a negative log. probability is a code length by information theory).

The algorithm design strategy is based on designing a heuristic procedure to find a single tree or set of trees that can be used to approximate Formula (1). This is described by the following 4 steps.

Step 1. Develop priors over the structural and continuous components of the model, $Pr(S \mid T)$ and $Pr(T)$. The form of the prior should be flexible enough so that it can be changed from application to application. In the IND package, these priors can be tailored to your application, and advice is given in the manual. Alternatively, "bland" priors can be used if you don't wish to assume a particular prior.

Step 2. Given a training sample *Sample*, determine a suitably efficient way of computing or approximating the posterior of the structural component of the model. Then devise a heuristic search procedure for searching the space of structures to find structures with high posterior. In trees, a simple one-ply look-ahead procedure can be tried, which corresponds to the standard tree growing algorithm [2]. In IND, two-ply and three-ply versions of look-ahead are also available. These start with the trivial, empty tree. They then consider extending the tree by a single ply, by replacing an ungrown node with a test and leaves at its outcomes. A heuristic measure to evaluate the quality of a new growth can be determined from the posterior probabilities. Several different tests are tried and evaluated, and the best one is chosen for subsequent development.

Step 3. Given a training sample *Sample* and a structure *T*, determine a formula or approximation for the posterior expected values of the parameters *S*, $Pr(new\text{-}class \mid new\text{-}case, T, Sample)$, as required for Formula~(1).

Step 4. Devise a procedure for approximating the summation of Formula (1) by a small set of high posterior structures. There are three techniques for doing this:

Smoothing: The sum can be computed in closed form if it is restricted to the set of tree structures obtained by pruning a large tree structure in all possible ways. A linear time algorithm is given in [6]. This is called smoothing because it is equivalent to smoothing out the class probabilities at the leaf of a tree by averaging them the branch leading to the leaf. This is implemented in the "-b" option to IND's `tprune`.

Averaging: The sum can be approximated by searching for and storing many dominant terms, i.e. many high posterior trees structures. We can build multiple tree structures, and combine them together efficiently in an AND-OR representation called *option trees*. Growing option trees and then applying a similar summation process to smoothing is called *tree averaging*. This is implemented as a style in IND's `mktree`.

Multiple Models: The sum can be approximated by using importance sampling or Monte Carlo estimation. That is, a few tree structures are generated in approximate proportion with their posterior (this is done using the tree growing heuristic), and their class probability vectors uniformly averaged.

PERFORMANCE SUMMARY

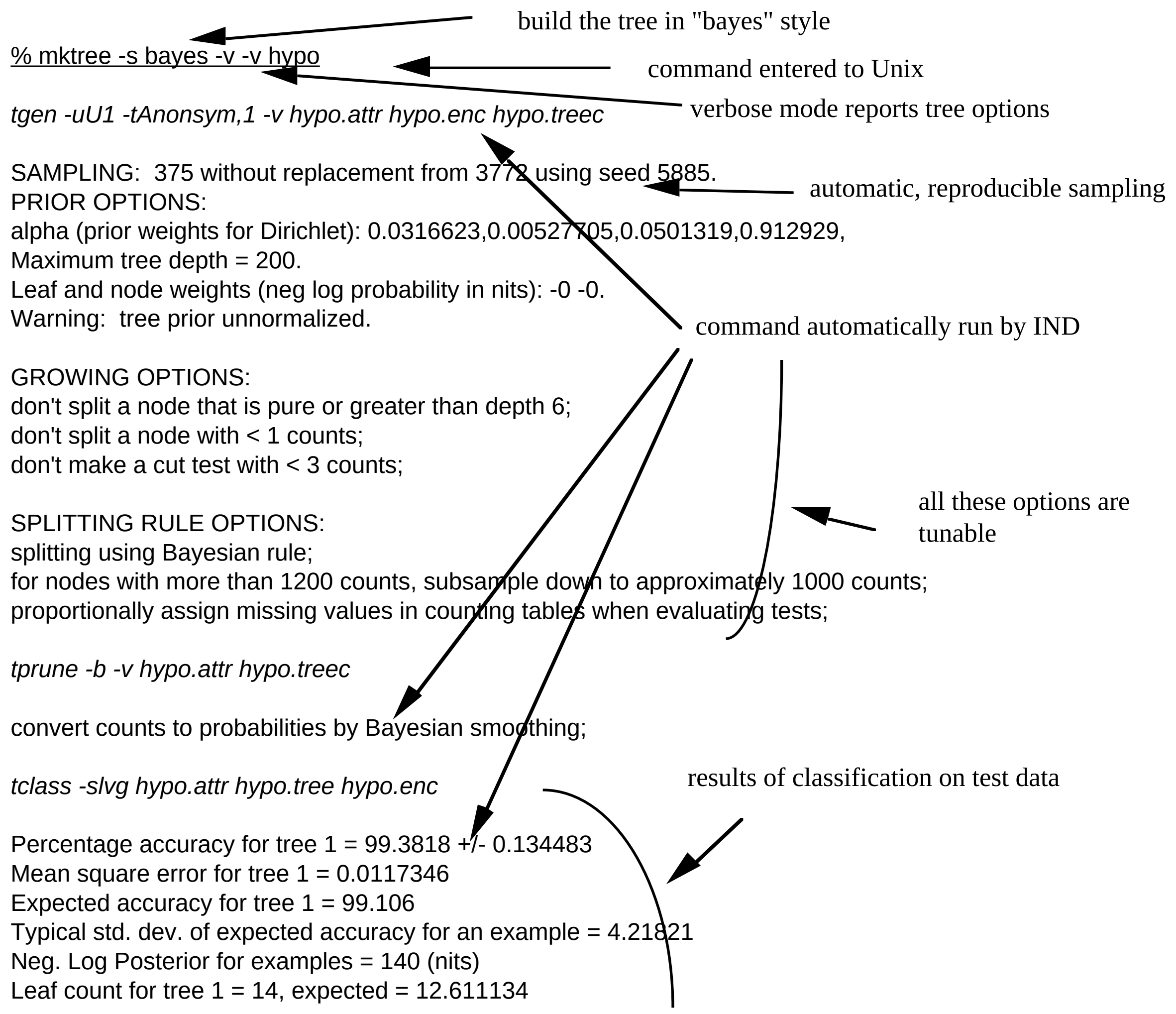
Various experimental results from the use of IND version 1.0 are reported in [6]. Experimental results for the graph component of IND, available in beta-test version 2.1, can be got from results in [7] for earlier code from Jon Oliver. IND has been run on databases available from University of California at Irvine (FTP to `ics.uci.edu` and look in the directory `machine-learning-databases`). The results show that the new features of IND give more accurate class probability estimates for new examples, and often better predictions, though sometimes at the cost of increased computation, depending on the problem. The MML graph component of IND has previously been run by Oliver and colleagues on DNA structure data and produced results of interest to molecular biologists, see [7] and references therein for details. IND has recently been hooked up to the System Diagnostic Builder from GHG Corporation, which is used for building diagnosis systems at NASA's Johnson Space Center [8]

Acknowledgements

IND was based on an early suite of software developed at Basser Department of Computer Science at Sydney University by a lineage of students of Jason Catlett: David Harper, Murray Dean, David Muller and Chris Carter, and possibly some others. More recently Rich Caruana of CMU and Jon Oliver of Monash University worked on the package during summer internship at NASA-Ames Research Center. Also the users of the beta-release provided considerable feedback.

References

[1] Berger, J. O. (1985). *Statistical Decision Theory and Bayesian Analysis*. Springer-Verlag, New York.
[2] Quinlan, J.R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81--106.
[3] Quinlan, J.R. (1992). *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA.
[4] Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees*. Wadsworth, Belmont.
[5] Buntine, W. (1991). Classifiers: A Theoretical and Empirical Study. *International Joint Conference on Artificial Intelligence*, August, 1991, Sydney.
[6] Buntine, W. (1992). Learning classification trees. *Statistics and Computing*, 2:63–73.
[7] Oliver, J. (1992). Inferring decision graphs using the minimum message length principle. *Australian Artificial Intelligence Conference*, November, 1992, Australia.
[8] Nieten, J. L. and Burke, R. (92). *System Diagnostic Builder*. Report from GHG Corporation at JSC.


build the tree in "bayes" style
command entered to Unix
verbose mode reports tree options
automatic, reproducible sampling
command automatically run by IND
all these options are tunable
results of classification on test data

```
% mktree -s bayes -v -v hypo

tgen -uU1 -tAnonsym,1 -v hypo.attr hypo.enc hypo.treec

SAMPLING:  375 without replacement from 3772 using seed 5885.
PRIOR OPTIONS:
alpha (prior weights for Dirichlet): 0.0316623,0.00527705,0.0501319,0.912929,
Maximum tree depth = 200.
Leaf and node weights (neg log probability in nits): -0 -0.
Warning:  tree prior unnormalized.

GROWING OPTIONS:
don't split a node that is pure or greater than depth 6;
don't split a node with < 1 counts;
don't make a cut test with < 3 counts;

SPLITTING RULE OPTIONS:
splitting using Bayesian rule;
for nodes with more than 1200 counts, subsample down to approximately 1000 counts;
proportionally assign missing values in counting tables when evaluating tests;

tprune -b -v hypo.attr hypo.treec

convert counts to probabilities by Bayesian smoothing;

tclass -slvg hypo.attr hypo.tree hypo.enc

Percentage accuracy for tree 1 = 99.3818 +/- 0.134483
Mean square error for tree 1 = 0.0117346
Expected accuracy for tree 1 = 99.106
Typical std. dev. of expected accuracy for an example = 4.21821
Neg. Log Posterior for examples = 140 (nits)
Leaf count for tree 1 = 14, expected = 12.611134
```

Figure 7. Building a tree using IND in verbose mode.

```
TSH < 6.05:  1.432e-05 0.0006207 2.909e-05 0.9993   negative
TSH >= 6.05:
|   TSH_measured = f:  0.0001371 4.283e-06 0.0002784 0.9996   negative
|   TSH_measured = t:
|   |   FTI < 64.5:
|   |   |   T4_measured = f:
|   |   |   |   on_thyroxine = f:
|   |   |   |   |   thyroid_surgery = f:  0.2523 9.126e-05 0.6358 0.1118   compensated_hypothyroid
|   |   |   |   |   thyroid_surgery = t:  0.04943 0.0004665 0.1141 0.836  negative
|   |   |   |   on_thyroxine = t:  0.01588 0.0003925 0.03357 0.9502   negative
|   |   |   T4_measured = t:
|   |   |   |   thyroid_surgery = f:  0.9637 1.226e-05 0.0007972 0.03548   primary_hypothyroid
|   |   |   |   thyroid_surgery = t:  0.08208 0.0001835 0.01192 0.9058   negative
|   |   FTI >= 64.5:
|   |   |   on_thyroxine = f:
|   |   |   |   TT4 < 150.5:
|   |   |   |   |   thyroid_surgery = f:  0.1531 8.899e-05 0.7433 0.1035   compensated_hypothyroid
|   |   |   |   |   thyroid_surgery = t:  0.00339 0.0001059 0.006886 0.9896   negative
|   |   |   |   TT4 >= 150.5:  0.04807 0.0001326 0.03691 0.9149   negative
|   |   |   on_thyroxine = t:  0.0004708 1.471e-05 0.0009563 0.9986   negative
```

Figure 8. A print of the resultant tree showing class probabilities and decisions.